When Will Ray-Tracing Replace Rasterization?

Kurt Akeley[*]
Stanford University
NVIDIA Corporation
Panelist

David Kirk[†]
NVIDIA Corporation
Panelist

Larry Seiler[‡]
ATI Research, Inc.
Panelist

Philipp Slusallek[§]
Saarland University
Panelist

Brad Grantham[¶]
Silicon Graphics Incorporated
Moderator

1 Motivation

Ray-tracing produces images of stunning quality but is difficult to make interactive. Rasterization is fast but making realistic images with it requires splicing many different algorithms together. Both GPU and CPU hardware grow faster each year. Increased GPU performance facilitates new techniques for interactive realism, including high polygon counts, multipass rendering, and texture-intensive techniques such as bumpmapping and shadows. On the other hand, increased CPU performance and dedicated ray-tracing hardware push the potential framerate of ray-tracing ever higher.

Depth-buffering rasterization made scanline geometric hidden-line techniques obsolete because it was easy to implement and dropping memory prices made implementations affordable, even though Sutherland, et al., concluded it was hopelessly inefficient.

2 Question for the Panel

Will the simplicity and/or increasing performance of ray-tracing make rasterization obsolete, and when? Speakers will address the future of rasterization versus ray-tracing/ray-casting techniques based on their broad and diverse industry experience and individual viewpoints as leaders in the graphics community.

3 Kurt Akeley

Rasterization hardware and the associated graphics standards, such as Direct 3D and OpenGL, will be around for a long time. Ray tracing will become feasible, outside of niche markets serviced with custom hardware, only as it can be implemented using rasterization hardware infrastructure. Because rasterization hardware is advancing in performance, flexibility, and programmability so rapidly, we can expect dramatic improvements in the image quality of interactive systems during the next few years. Applications will introduce ray tracing algorithms gradually, synthesizing rasterization, ray tracing, and other global shading techniques to obtain the best overall results.

Kurt Akeley works part time at NVIDIA Corporation, where he is a member of the graphics architecture team. He spends the rest of his week at Stanford, working toward the completion of the electrical engineering Ph.D. that he put on hold in 1982 to co-found Silicon Graphics.

During his 19 years at Silicon Graphics Kurt lead the development of several high-end graphics systems, including GTX, VGX, and RealityEngine. He also lead the development of OpenGL, and he continues to be involved with OpenGL's evolution. Kurt's past SIGGRAPH participation includes course presentations, panel presentations , and participation in the technical program, which he chaired in 2000. Kurt is named on thirteen patents, is a fellow of the ACM, and in 1995 was the recipient of the ACM SIGGRAPH Computer Graphics Achievement Award.

4 David Kirk

Ray Tracing vs. Rasterization is an interesting question, although perhaps a bit ill-posed. Ray tracing is used both as a visibility technique and a catch-all to describe a variety of global illumination techniques. Usually these techniques are CPU-based. Rasterization, typically depth buffering, is primarily used as a visibility technique, although some texture-based shadowing, illumination, and reflection techniques can make use of rasterization as the workhorse. Usually these techniques are GPU-based.

I'll be interested in discussing a bigger question, though: "When will hardware graphics pipelines become sufficiently programmable to efficiently implement ray tracing and other global illumination techniques?". I believe that the answer is now, and more so from now on! As GPUs become increasingly programmable, the variety of algorithms that can be mapped onto the computing substrate of a GPU becomes ever broader. As part of this quest, I routinely ask artists and programmers at movie and special effects studios what features and flexibility they will need to do their rendering on GPUs, and they say that they could never render on hardware! What do they use now: crayons? Actually, they use hardware now, in the form of programmable general-purpose CPUs. I believe that the future convergence of realistic and real-time rendering lies in highly programmable special-purpose GPUs.

David is Chief Scientist and Vice President of Architecture at NVIDIA. He was previously Chief Scientist and head of technology, developing video game software for Crystal Dynamics, and prior to that worked on developing graphics hardware for engineering workstations at Apollo/Hewlett-Packard. David holds B.S. and M.S. degrees from MIT and M.S. and Ph.D. degrees from the California Institute of Technology, and is the author/inventor of over 100 technical publications and patents in the area of computer graphics and hardware.

[*]e-mail: kurt_akeley@acm.org
[†]e-mail: davidk@nvidia.com
[‡]e-mail: lseiler@ati.com
[§]e-mail: slusallek@cs.uni-sb.de
[¶]e-mail: grantham@sgi.com

5 Larry Seiler

Ray-tracing will never replace rasterization. Both will be subsumed by something that is better than either, for three reasons. First, the two algorithms do not solve the same problem. Ray-tracing excels at modeling a physical environment. Rasterization excels at splicing many different algorithms together (that's a strength, not a weakness) to achieve a desired result that may not match physical reality. Second, memory bandwidth doesn't increase as fast as CPU performance. Rasterization algorithms are better able to produce the reference locality and good cache behavior that is required to use memory bandwidth efficiently. Finally, rasterization is typically better at tolerating long latencies, which are getting worse as GPUs and graphics models get more complicated.

The Next Big Thing needs to have the strengths of both algorithms and needs to resolve a problem common to both: aliasing! Standard rasterization and standard ray-tracing algorithms create point samples that must be post-filtered, instead of correctly pre-filtering the model. They also share the inefficiency of repeating similar operations many times when rendering multiple frames. Possible futures are suggested by work on beam-casting, IBR, and by the Shadermaps algorithm, which saves intermediate computations at multiple resolution levels for use in multiple frames. These and other techniques should be extended to allow mixing ray-traced, rasterized, and even volumetric techniques, probably implemented in ways that blur the distinction between ray-tracing and rasterization.

Larry Seiler is a hardware architect at ATI Research, Inc. Previously, Larry worked at Real Time Visualization developing volumetric ray tracing algorithms. Larry described this work at the 1999 MicroProcessor Forum and is a co-author of "The VolumePro Real-Time Ray-casting System" (SIGGRAPH 1999). Other paper credits include "Quadratic Interpolation for Near-Phong Quality Shading" (SIGGRAPH 1998 technical sketch) and co-authorship of "Neon: A Single-Chip 3D Workstation Graphics Accelerator" (1998 EUROGRAPHICS / SIGGRAPH Workshop on Graphics Hardware, best paper). Larry holds B.S. and M.S. degrees from Caltech and a Ph.D. degree from MIT. Larry is an inventor for 19 issued patents.

6 Philipp Slusallek

The ray-tracing algorithm does have a number of key benefits that make it significantly more attractive to use than the rasterization algorithm of todays graphics hardware:

- Flexibility: Ray-tracing can accurately compute exactly those rays required for some effect, while rasterization is highly limited in what it can compute efficiently.

- Generality: Because of this flexibility, ray-tracing is the perfect match for many advanced rendering algorithms and effects required for future 3D graphics.

- Approximations: Rasterization has to rely on approximations and special hacks for almost every rendering effect. These hacks are usually non-intuitive, non-trivial to find and implement, and are costly or even impossible to combine with other approximations.

- Plug & Play: Ray-tracing directly uses high-level shaders (such as RenderMan) that can be combined with arbitrary complexity. Even large scenes with many, complex shaders simply work as expected without further work or tweaking of parameters.

- Scalability: Ray-tracing scales extremely well both in scene size (logarithmic versus linear for rasterization) and in terms of the number of processing elements it can make use of. The latter issue will be important for future hardware where billions of transistors in a chip must be kept busy.

Due to these arguments I am convinced that in the future, 3D graphics will be based on ray-tracing. The biggest issue along this road is the question about the best API and the transition to it.

What remains is the question about hardware: Will ray-tracing and 3D graphics run on the GPU, CPU, or on some new special-purpose hardware? The stream-processing model of the GPU has many arguments in its favor and will most likely support some form of ray-tracing in the near future. However the CPU guys are not standing still either and will likely support high-performance ray-tracing on single CPUs in the not too distant future, too.

It will be interesting to watch the two groups, each striving to build the best hardware support for ray-tracing and the future of 3D graphics.

Philipp is full professor at the computer graphics lab of Saarland University. From 1998 to 1999 he was visiting assistant professor at the Stanford University graphics lab. He received a Diploma in physics from the University of Tubingen, Germany in 1990 and a PhD in computer science from the University of Erlangen, Germany in 1995. He was responsible for the design of a commercial 2D/3D CAD package and has been leading the Vision project, a large object-oriented and physically-based rendering system. His current research activities focus on interactive ray-tracing and interactive lighting simulation. Other research topics include design of a network-integrated multi-media infrastructure, consistent illumination in virtual environments, physically-based and realistic image synthesis, and object-oriented software design. Philipp organized the SIGGRAPH 2001 course "Interactive Ray-Tracing".

7 Brad Grantham

Rasterization has always represented speed because of its direct forward mapping from triangles to screen pixels. Over its many years of dominance in interactive rendering, lots of techniques have been incorporated for increased realism. Ray-tracing, however, provides global illumination and high-quality surface appearance at a much lower price in implementation. Ray-tracing will continue to increase in speed as Moore's law carries us forward and as commodity ray-tracing hardware appears, but rasterization relies on the creativity and cleverness of engineers and artists in the face of exponentially increasing complexity. Perhaps some new algorithm will triumph over both in the long term but my prediction consists of the adoption within five to ten years of ray-tracing over rasterization (with perhaps an intermediate hybrid) through a dedicated device much like today's AGP 4x graphics adapter and an open standard API embodying both ease-of-use and performance much as OpenGL has provided for rasterization.

Brad Grantham works in the Applications Engineering division at SGI, specializing in helping application developers use OpenGL and SGI's toolkits for visualization. Previously, Brad worked for VA Linux Systems on graphics workstation development and before that in the Advanced Graphics Software division of Silicon Graphics, Inc. on scene graphs and toolkits.

Brad spoke at the "Advanced Rendering Techniques in OpenGL" course at SIGGRAPH from 1997 through 2000 and organized the "OpenGL on Linux" course at the Fall 2000 Game Developers' Conference. Brad also designed and implemented the open source "ACTC" API and library for creating triangle strips from triangles.